JAVA

Created by

James Gosling

LEARNINGVERSE

-Happy Learning

What is Java?

Java is a popular programming language, created in 1995.
It is owned by Oracle, and more than **3 billion** devices run Java.
It is used for:

- Mobile applications (specially Android apps)
- Desktop applications
- Web applications
- Web servers and application servers
- Games
- Database connection

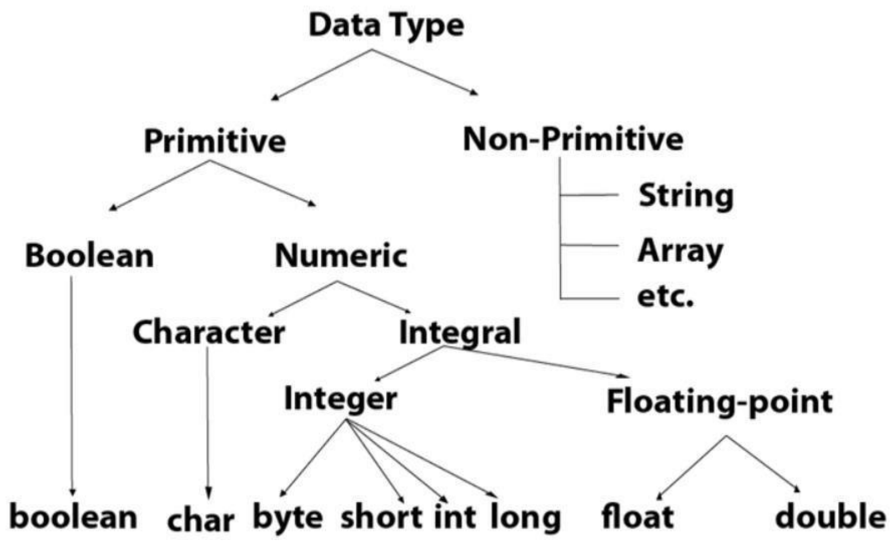
And much, much more!

Why Use Java?

- Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
- It is one of the most popular programming language in the world
- It has a large demand in the current job market
- It is easy to learn and simple to use
- It is open-source and free
- It is secure, fast and powerful
- It has a huge community support (tens of millions of developers)
- Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
- As Java is close to C++ and C#, it makes it easy for programmers to switch to Java or vice versa

Java Datatypes

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.



```
public class Main {
 public static void main(String[] args) {
   int myNum = 5;               // integer (whole number)
   float myFloatNum = 5.99f;    // floating point number
   char myLetter = 'D';         // character
   boolean myBool = true;       // boolean
   String myText = "Hello";     // String
   System.out.println(myNum); //5
   System.out.println(myFloatNum);//5.9
   System.out.println(myLetter);//D
   System.out.println(myBool);//true
   System.out.println(myText);//Hello
 }
}
```

Control Statements

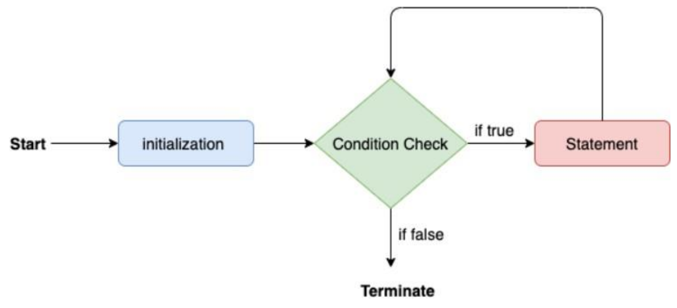
1. Decision Making statements
    - if statements
    - switch statement
2. Loop statements
    - do while loop
    - while loop
    - for loop
    - for each loop

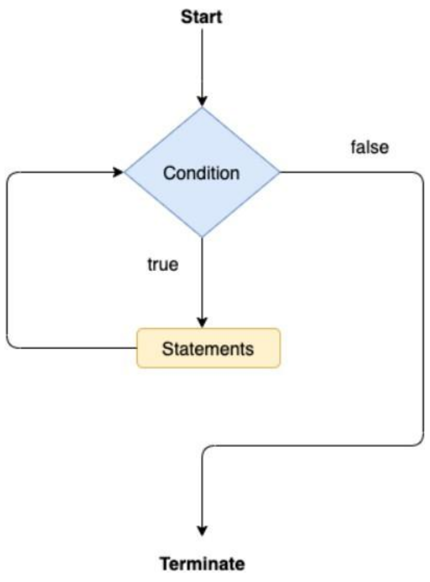
3.Jump statements
- break statement
- continue statement

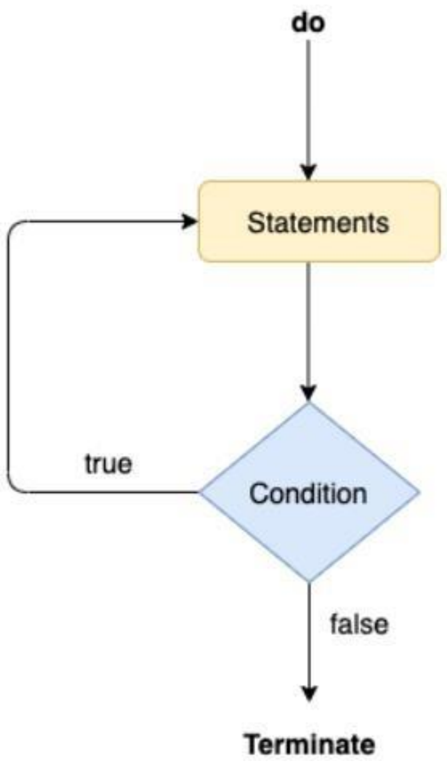
Example:

```
public class Student {
public static void main(String[] args) {
String fruit = "Apple";
if(fruit == "Apple") {
System.out.println("fruit is apple");
}else if (fruit == "Banana") {
System.out.println("fruit is banana");
}else if(fruit == "Orange") {
System.out.println("fruit is orange");
}else {
System.out.println(fruit);
}
}
}
```

Loops:


Start
initialization
Condition Check
if true
Statement
if false
Terminate


Start
Condition
false
true
Statements
Terminate


do
Statements
true
Condition
false
Terminate

Example:

```
public class Main {

  public static void main(String[] args) {

    int i = 0;

    while (i < 5) {

    System.out.println(i);

    i++;

    }

  }
}

    public class Main {

  public static void main(String[] args) {

    for (int i = 0; i < 5; i++) {

      System.out.println("LearningVerse");

    }

  }

}
```

Java Methods:

To call a method in Java, write the method's name followed by two parentheses **()** and a semicolon**;**
In the following example, myMethod() is used to print a text (the action), when it is called: Let’s see an example:

```
public class Main {
static void myMethod() {
System.out.println("Happy Learning");
}
public static void main(String[] args) {myMethod();
myMethod();myMethod();
}
}
```

Java Method Parameters:

Information can be passed to methods as parameter. Parameters act as variables inside the method.
Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.
Example:

```
public class Main {
static void myMethod(String fname) {
System.out.println(fname + " Thomas");
}
public static void main(String[] args) {myMethod("Geetha");
myMethod("Shaju");
myMethod("Roshan");
}
}
```

Java Method Overloading:

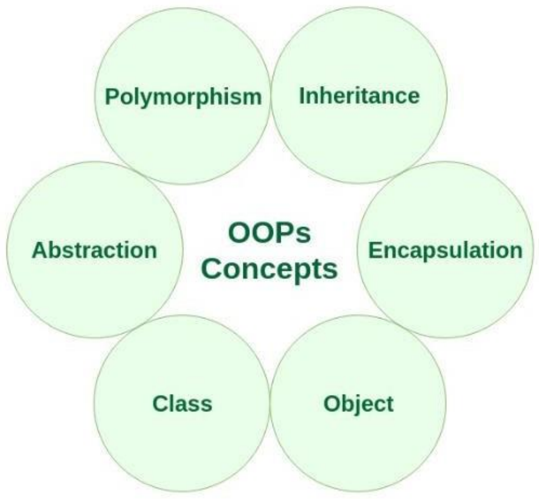
With method overloading**,** multiple methods can have the same name with different parameters. Let us see an example:

```
public class Main {
static int plusMethodInt(int x, int y) {return x + y;
}
static double plusMethodDouble(double x, double y) {return x +
y;
}
public static void main(String[] args) {int myNum1 =
plusMethodInt(8, 5);
double myNum2 = plusMethodDouble(4.3, 6.26);
System.out.println("int: " + myNum1);
System.out.println("double: " + myNum2);
}
}
```

```
public class Main {
static int plusMethodInt(int x, int y) {return x + y;
}
static double plusMethodDouble(double x, double y) {return x +
y;
}
public static void main(String[] args) {int myNum1 =
plusMethodInt(8, 5);
double myNum2 = plusMethodDouble(4.3, 6.26);
System.out.println("int: " + myNum1);
System.out.println("double: " + myNum2);
}
}
```

Java OPP:

OOP stands for **Object-Oriented Programming**.



Java Classes/Objects

Java is an object-oriented programming language.Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.

A Class is like an object constructor, or a "blueprint" for creating objects.

```
public class Main { int x = 5;
public static void main(String[] args) {
        Main myObj = new Main();
        System.out.println(myObj.x);
}
```

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Java ArrayList:

The ArrayList class is a resizable array, which can be found in the **java.util package.**

The ArrayList class has many useful methods. For example, to add elements to the ArrayList, use the add() method:

Example:

```
import java.util.ArrayList;
public class Main {
        public static void main(String[] args) {
                ArrayList<String> cars = new ArrayList<String>();
                cars.add("Volvo");
                cars.add("BMW");
                cars.add("Ford");
                cars.add("Mazda"); System.out.println(cars);
        }
}
```

To access an element in the ArrayList, use the get() method and refer to the index number:

Cars.get(0);

To modify an element, use the set() method and refer to the index number.

To remove an element, use the remove() method and refer to the index number.

To remove all the elements in the ArrayList, usethe clear() method.

To find out how many elements an ArrayList have,use the size method.

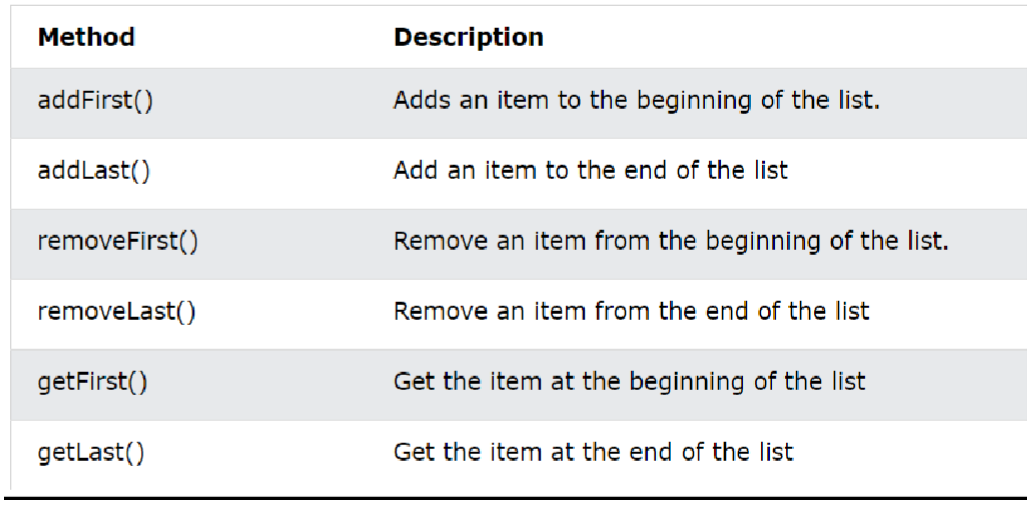
Loop through the elements of an ArrayList with a for loop, and use the size() method to specify how many times the loop should run.

Java LinkedList:

The LinkedList class is almost identical to the ArrayList.import java.util.LinkedList;

```
public class Main {
	public static void main(String[] args) {
		LinkedList<String> cars = new LinkedList<String>();
		cars.add("Volvo");
		cars.add("BMW");
		cars.add("Ford");
		cars.add("Mazda");
		System.out.println(cars);
	}
}
```

LinkedList Methods



Method	Description
addFirst()	Adds an item to the beginning of the list.
addLast()	Add an item to the end of the list
removeFirst()	Remove an item from the beginning of the list.
removeLast()	Remove an item from the end of the list
getFirst()	Get the item at the beginning of the list
getLast()	Get the item at the end of the list

Java HashMap:

A HashMap however, store items in "**key/value**" pairs, and you can access them by an index of another type (e.g. a String).

One object is used as a key (index) to another object (value). It can store different types: String keys and Integer values, or the same type, like: String keys and String values:

The HashMap class has many useful methods. For example, to add items to it, use the put() method.

```
import java.util.HashMap;

public class Main {

  public static void main(String[] args) {

    HashMap<String, String> capitalCities = new
    HashMap<String, String>();

    capitalCities.put("England", "London");

    capitalCities.put("Germany", "Berlin");

    capitalCities.put("Norway", "Oslo");

    capitalCities.put("USA", "Washington DC");

    System.out.println(capitalCities);

  }

}
```

To access a value in the HashMap, use the get() method and refer to its key.

To remove an item, use the remove() method and refer to the key.

To remove all items, use the clear() method.

To find out how many items there are, use the size() method.

Loop through the items of a HashMap with a **for-each** loop.

Note: Use the keySet() method if you only want the keys, and use the values() method if you only want the values.

Java Exceptions:

When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.
When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an **exception** (throw an error).

Java try and catch

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs:

Finally

The finally statement lets you execute code, after try...catch, regardless of the result:

```
public class Main {

  public static void main(String[] args) {

    try {

      int[] myNumbers = {1, 2, 3};

      System.out.println(myNumbers[10]);

    } catch (Exception e) {
```

```
    System.out.println("Something went wrong.");
    } finally {
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

The throw keyword

The throw statement allows you to create a custom error.

The throw statement is used together with an **exception type**. There are many exception types available in Java: ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc: